Compte rendu système

Communication par sockets

[Objectif 3](#_Toc533180354)

[Fonctions utilisées 3](#_Toc533180355)

[Commun client/serveur 3](#_Toc533180356)

[Création de socket 3](#_Toc533180357)

[Connection : sockaddr / sockaddr\_in 3](#_Toc533180358)

[Sockaddr\_in : init 4](#_Toc533180359)

[Sockaddr\_in : htons / inet\_addr 4](#_Toc533180360)

[Coté client 4](#_Toc533180361)

[Connection : connect 4](#_Toc533180362)

[Coté serveur 5](#_Toc533180363)

[Liaison de la socket serveur : bind 5](#_Toc533180364)

[Socket en mode attente passive : listen 5](#_Toc533180365)

[Réception de la connexion d’un client : accept 5](#_Toc533180366)

[Réception / envoi de messages 5](#_Toc533180367)

[Read / write 5](#_Toc533180368)

[Recv / send 6](#_Toc533180369)

[Recvto / sendto 7](#_Toc533180370)

[Read/write ou send/recv 7](#_Toc533180371)

[Application fournie 7](#_Toc533180372)

[Options 7](#_Toc533180373)

[Le client 7](#_Toc533180374)

[Le serveur 8](#_Toc533180375)

[Fonctionnalités 8](#_Toc533180376)

[Les commandes 8](#_Toc533180377)

[Conclusion 8](#_Toc533180378)

# *Objectif*

Réaliser une communication client / serveur, le serveur devant être capable de gérer plusieurs clients.

# Fonctions utilisées

## Commun client/serveur

### Création de socket

Int socket(int domain, int type, int protocol)

Domain : famille de protocole à utiliser (af\_inet : ipv4)

Type : sémantique de la communication (sock\_stream : tcp)

Protocol : protocol à utiliser (0 : par défaut)

### Connection : sockaddr / sockaddr\_in

Identique du point de vu mémoire :

* On passe de l’un a l’autre par cast

Sockaddr\_in est plus parlant du point de vu des variables :

|  |  |
| --- | --- |
| **Sockaddr\_in** | **Sockaddr** |
|  |  |

### Sockaddr\_in : init

Sin\_family : socket domain (af\_inet)

Sin\_port : port de connection (htons(port))

Sin\_addr : addresse de connection (inet\_addr(127.0.0.1) | inaddr\_any : toutes les interfaces locales)

### Sockaddr\_in : htons / inet\_addr

Htons : convertit l’ordre de bit hôte en ordre de bit réseau

Inet\_addr : convertit l’adresse de la notation ipv4 pointée en binaire ordonné réseau

## Coté client

### Connection : connect

Int connect(int sock, sockaddr\* sin, socklen\_t taddr)

Sock : socket à connecter

Sin : ici on cast la sockaddr\_in : (struct sockaddr\*)&sin

Taddr : taille de l’adresse : sizeof(sin)

## Coté serveur

### Liaison de la socket serveur : bind

Int bind(int sock, sockaddr\* sin, socklen\_t taddr)

Sock : socket à connecter

Sin : ici on cast la sockaddr\_in : (struct sockaddr\*)&sin

Taddr : taille de l’adresse : sizeof(sin)

### Socket en mode attente passive : listen

Int listen(int sock, int backlog)

Sock : socket à passer en attente passive

Backlog : taille de la file d’attente pour la connexion

### Réception de la connexion d’un client : accept

Int accept(int sock, sockaddr\* sin, socklen\_t\* taddr)

Sock : socket passive du serveur

Sin : ici on cast la sockaddr\_in : (struct sockaddr\*)&sin

Taddr : taille de l’adresse : sizeof(sin)

**Retour** : socket en communication avec le client connecté

## Réception / envoi de messages

### Read / write

Read:

Int read(int sock, char\* buffer, size\_t count)

Sock : socket avec laquelle on communique

Buffer : buffer de lecture

Count : taille du message à réceptionner.

Write:

Int write(int sock, char\* buffer, size\_t count)

Sock : socket avec laquelle on communique

Buffer : buffer d’écriture

Count : taille du message à envoyer

### Recv / send

Recv

Recv(int sock, char\* buffer, size\_t count, int flags)

Sock : socket avec laquelle on communique

Buffer : buffer de lecture

Count : taille du message à recevoir

Flags : options de reception

Send

Send(int sock, char\* buffer, size\_t count, int flags)

Sock : socket avec laquelle on communique

Buffer : buffer d’écriture

Count : taille du message à envoyer

Flags : options d’envoi

### Recvto / sendto

S’utilise comme recv et send mais permet d’envoyer/de recevoir des messages sur une scket qui n’est pas connectée entre le client et le serveur.

### Read/write ou send/recv

Send/recv permet de spécifier plus d’options grâce aux flags (voir doc)

# Application fournie

### Options

-p : numéro du port

-a : adresse en notation pointée (client seulement)

### Le client

Le client se connecte automatiquement au serveur référencé par l’adresse et le port donnés. Affiche ce qu’il reçoit du serveur, récupère une réponse de l’utilisateur et l’envoie au serveur.

### Le serveur

Le serveur s’initialise et se met en attente de demandes de connexions, quand une demande est détectée, on lance l’application pour ce client dans un processus fils. Le serveur envoie une chaine de caractères, reçoit une réponse du client puis traite celle-ci.

### Fonctionnalités

Si le client tape une chaine de caractères quelconque et que l’application serveur n’en attends pas une, le serveur lui renvoie la chaine tapée.

Si le client envoie une chaine du type « /commande », la chaine va être traitée en tant que commande, lancer un service spécialisé si la commande est reconnue ou signaler au client que la commande est inconnue.

### Les commandes

/exit pour déconnecter le client

/reverse qui lance un service d’inversion de chaine de caractères

# Conclusion

Au cours de se tp, nous avons appris à utiliser la communication par sockets, à gérer la connexion de plusieurs clients à un serveur grâce à la création de processus ainsi que la gestion des erreurs en c.

Des difficultés ont été rencontrée sur la machine virtuelle qui n’exécutait pas le code correctement sur ma machine personnelle (le code bloquait sans raisons)